НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

КАФЕДРА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

**Лабораторна робота №7**

з дисципліни **«**Основи технологій програмування**»**

Варіант: «6412»

Виконав:

студент 2 курсу

група ІП-64

Лейзьо Сергій Іванович

Перевірив:

Подрубайло Олександр Олександрович

Київ – 2018

**Завдання**

**Тема:** Обробка виключних ситуацій та основи тестування в мові програмування Java.

**Мета:** Здобуття навичок у використанні механізму обробки виключних ситуацій та написанні тестів для перевірки працездатності методів в мові

програмування Java.

**1.** Модифікувати класи з попередніх лабораторних робіт (лабораторні роботи №5та №6) таким чином, щоб обробка виключних ситуацій відбувалась за

допомогою стандартних засобів мови програмування Java. Створити власний

клас обробник виключних ситуацій.

**2.** Написано JUnit-тести для перевірки працездатності усіх методів та виключних ситуацій.

**3.** Всі початкові дані задаються у виконавчому методі. Код повинен відповідати

стандартам JCC .

**Програмний код**

**Cucumber**

package com.company;  
  
public class Cucumber extends Vegetable{  
 public Cucumber(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность огурца: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес огурца: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
 }

**Feta**

package com.company;  
  
public class Feta extends Vegetable{  
 public Feta(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность феты: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес феты: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**IllegalCaloricityException**

package com.company;  
  
public class IllegalCaloricityException extends IllegalArgumentException{  
 public IllegalCaloricityException(String s) {  
 super(s);  
 }  
}

**IllegalWeightException**

package com.company;  
  
public class IllegalWeightException extends IllegalArgumentException {  
 public IllegalWeightException(String s) {  
 super(s);  
 }  
}

**Main**

package com.company;  
import java.util.ArrayList;  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 /\*  
 Salad salad=new Salad();  
 salad.Salad();  
 System.out.println("\nKaлорийность салата: " + salad.CalSalad());  
 System.out.println("Как вы хотите отсортировать овощи? 1-по калориям 2-по массе");  
 Scanner sort=new Scanner(System.in);  
 int number=sort.nextInt();  
 salad.Sort(number);  
 System.out.println("Введите диапазон калорийности от a до b:");  
 System.out.print("a:");  
 Scanner from=new Scanner(System.in);  
 int a=from.nextInt();  
 System.out.print("b:");  
 Scanner to=new Scanner(System.in);  
 int b=to.nextInt();  
 salad.Search(a,b);  
  
 VegetableList vegetableList=new VegetableList();  
 Vegetable a=new Cucumber(1,1,"a");  
 Vegetable b=new Cucumber(2,2,"b");  
 Vegetable c=new Cucumber(3,3,"c");  
 Vegetable d=new Cucumber(3,3,"c");  
 vegetableList.add(a);  
 vegetableList.add(b);  
 vegetableList.add(c);  
 //System.out.println(vegetableList.get(0));  
 // System.out.println(vegetableList.size());  
 // System.out.println(vegetableList.isEmpty());  
 // System.out.println(vegetableList.contains(d));  
 // Vegetable list2[] = new Vegetable[vegetableList.size()];  
 //list2 = vegetableList.toArray(list2);  
 //for(Vegetable number : list2){  
 // System.out.println("Number = " + number);  
 // }  
 //vegetableList.remove(1);  
 //System.out.println(vegetableList.get(1));  
 ArrayList<Vegetable> deliveryCities = new ArrayList<>();  
  
 deliveryCities.add(d);  
 deliveryCities.add(c);  
 vegetableList.addAll(deliveryCities);  
 System.out.println(vegetableList.get(4));  
 \*/  
 }  
}

**Onion**

package com.company;  
  
public class Onion extends Vegetable{  
 public Onion(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность лука: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес лука: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**RedPepper**

package com.company;  
  
public class RedPepper extends Vegetable{  
 public RedPepper(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность красного перца: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес красного перца: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**Salad**

package com.company;  
  
import java.util.Arrays;  
import java.util.Comparator;  
  
public class Salad{  
 Vegetable param[]={new Cucumber(16,100,"Cucumber"),  
 new RedPepper(20,110,"RedPepper"),  
 new Tomato(180,101,"Tomato"),  
 new Feta(264,200,"Feta"),  
 new Onion(35,90,"Onion")};  
 public void Salad(){  
 for(int i=0;i<param.length;i++){  
 this.param[i].Weight();  
 this.param[i].Caloricity();  
 }  
 }  
 public double CalSalad(){  
 double cal=0;  
 for(int i=0;i<param.length;i++){  
 cal=cal+param[i].caloricity;  
 }  
 return cal;  
 }  
 public void Sort(int number)  
 {  
 switch (number){  
 case 1: Arrays.*sort*(param, Comparator.*comparingDouble*(o -> o.getCaloricity()));  
 System.*out*.println("Отсортировано по калориям: ");  
 for(int i=0;i<param.length;i++){  
 System.*out*.print(param[i].getCaloricity()+" ");  
 System.*out*.println(param[i].toString());  
 }  
 break;  
 case 2: Arrays.*sort*(param, Comparator.*comparingDouble*(o -> o.getWeight()));  
 System.*out*.println("Отсортировано по весу: ");  
 for(int i=0;i<param.length;i++){  
 System.*out*.print(param[i].getWeight()+" ");  
 System.*out*.println(param[i].toString());  
 }  
 break;  
 default: System.*out*.println("Вы ввели не то число!");  
 }  
 }  
 public void Search(int a,int b){  
 System.*out*.println("В этот диапазон входят: ");  
 for(int i=0;i<param.length;i++){  
 if((param[i].getCaloricity()>=a)&&(param[i].getCaloricity()<=b)){  
 System.*out*.println(param[i].toString());  
 }  
 }  
 }  
}

**Tomato**

package com.company;  
  
public class Tomato extends Vegetable{  
 public Tomato(double caloricity, double weight, String vegetables){  
 super(caloricity,weight,vegetables);  
 }  
 @Override  
 public void Caloricity(){  
 System.*out*.println(" Калорийность помидора: "+ super.caloricity);  
 }  
 @Override  
 public void Weight(){  
 System.*out*.print("Вес помидора: "+super.weight);  
 }  
 @Override  
 public double getWeight(){  
 return super.weight;  
 }  
 @Override  
 public double getCaloricity(){  
 return super.caloricity;  
 }  
 @Override  
 public String toString() {  
 return super.vegetables;  
 }  
}

**Vegetable**

package com.company;  
  
abstract class Vegetable{  
 protected double caloricity;  
 protected double weight;  
 protected String vegetables;  
 abstract public void Caloricity();  
 abstract public void Weight();  
 abstract public double getWeight();  
 abstract public double getCaloricity();  
 abstract public String toString();  
  
 public Vegetable(double caloricity, double weight, String vegetables) {  
 if (caloricity <= 0) throw new IllegalCaloricityException("This product impossible!");  
 if (weight <= 0) throw new IllegalWeightException("Weight must be >0!");  
 this.caloricity = caloricity;  
 this.weight = weight;  
 this.vegetables = vegetables;  
 }  
}

**VegetableList**

package com.company;  
  
import java.util.Collection;  
import java.util.Iterator;  
import java.util.List;  
import java.lang.reflect.Array;  
import java.util.ListIterator;  
  
public class VegetableList implements List<Vegetable> {  
 private int size=0; //размер List  
  
 private ListNode head;  
 private ListNode tail;  
  
 //Пустой конструктор  
 public VegetableList(){  
 }  
 public VegetableList(Vegetable vegetable){  
 add(vegetable);  
 }  
 public VegetableList(Collection<? extends Vegetable> c){  
 addAll(c);  
 }  
  
 //Класс для хранения узлов  
 private class ListNode {  
 private Vegetable vegetable;  
 private ListNode next;  
 private ListNode prev;  
  
 private ListNode(Vegetable vegetable) {  
 this.vegetable = vegetable;  
 }  
  
 private ListNode(Vegetable vegetable, ListNode next, ListNode prev) {  
 this.vegetable=vegetable;  
 this.next = next;  
 this.prev = prev;  
 }  
 }  
 //Возвразает размер VegetableList  
 @Override  
 public int size() {  
 return size;  
 }  
 //Возвращает пуста ли коллекция  
 @Override  
 public boolean isEmpty() {  
 return size==0;  
 }  
 //Возвращает true, если этот список содержит указанный элемент  
 @Override  
 public boolean contains(Object o) {  
 if (head == null) {  
 return false;  
 }  
 for (ListNode data = head; data != null; data = data.next) {  
 if (o.equals(data.vegetable)) {  
 return true;  
 }  
 }  
 return false;  
 }  
  
 @Override  
 public Iterator<Vegetable> iterator() {  
 return new Iterator<>() {  
 ListNode current = head;  
  
 @Override  
 public boolean hasNext() {  
 return current != null;  
 }  
  
 @Override  
 public Vegetable next() {  
 if(hasNext()){  
 Vegetable data = current.vegetable;  
 current = current.next;  
 return data;  
 }  
 return null;  
 }  
  
 @Override  
 public void remove(){  
 throw new UnsupportedOperationException ("Remove not implemented.");  
 }  
 };  
 }  
  
 @Override  
 public Object[] toArray() {  
 Object[] result = new Object[size];  
 int number = 0;  
 for (ListNode data = head; data!= null; data = data.next)  
 result[number++] = data.vegetable;  
 return result;  
 }  
 //Возвращает массив, содержащий все элементы в этом списке  
 @SuppressWarnings("unchecked")  
 @Override  
 public <T> T[] toArray(T[] array) {  
 if (array.length < size)  
  
 array = (T[]) Array.*newInstance*(  
 array.getClass().getComponentType(), size);  
 int number = 0;  
 Object[] result = array;  
 for (ListNode data = head; data != null; data = data.next)  
 result[number++] = data.vegetable;  
  
 if (array.length > size) {  
 array[size] = null;  
 }  
 return array;  
 }  
 //Добавляет элемент в список  
 @Override  
 public boolean add(Vegetable vegetable) {  
 //Добавление в конец списка  
 ListNode newNode = new ListNode(vegetable);  
 if(head == null){  
 head = newNode;  
 tail = newNode;  
 }  
 else{  
 newNode.prev = tail;  
 tail.next = newNode;  
 tail = newNode;  
 }  
 size++;  
 /\*  
 //Добавление в начало списка  
 ListNode newNode = new ListNode(vegetable);  
 if(head == null){  
 head = newNode;  
 tail = newNode;  
 }  
 else{  
 newNode.next = head;  
 head.prev = newNode;  
 head = newNode;  
 }  
 size++;\*/  
 return true;  
 }  
 //Удаляет элемент, возвращая его содержимое  
 @Override  
 public boolean remove(Object o) {  
 if (head == null) {  
 return false;  
 }  
 for (ListNode data = head; data != null; data = data.next) {  
 if (o.equals(data.vegetable)) {  
 unlink(data);  
 return true;  
 }  
 }  
 return false;  
 }  
 private void unlink(ListNode data) {  
 final ListNode next = data.next;  
 final ListNode prev = data.prev;  
  
 if (prev == null) {  
 head = next;  
 } else {  
 prev.next = next;  
 data.prev = null;  
 }  
  
 if (next == null) {  
 tail = prev;  
 } else {  
 next.prev = prev;  
 data.next = null;  
 }  
 data.vegetable = null;  
 size--;  
 }  
 //Возвращает, содержатся ли все элементы c в коллекции  
 @Override  
 public boolean containsAll(Collection<?> c) {  
 for (Object o : c)  
 if (contains(o)) {  
 return true;  
 }  
 return false;  
 }  
 //Добавляет все элементы в указанной коллекции до конца  
 @Override  
 public boolean addAll(Collection<? extends Vegetable> c) {  
 for (Vegetable vegetable : c) {  
 add(vegetable);  
 }  
 return true;  
 }  
 //Добавляет все элементы в указанной коллекции до конца начиная с index  
 @Override  
 public boolean addAll(int index, Collection<? extends Vegetable> c) {  
 if (index == size) {  
 addAll(c);  
 return true;  
 }  
 if (c.isEmpty()) {  
 return false;  
 }  
 VegetableList listAdd = new VegetableList(c);  
 final ListNode newHead = listAdd.head;  
 final ListNode newTail = listAdd.tail;  
 if (index == 0) {  
 newTail.next = head;  
 newHead.prev = head.prev;  
 head.prev = newTail;  
 head = newHead;  
 } else {  
 ListNode temp = getElement(index);  
 final ListNode prev = temp.prev;  
  
 prev.next = newHead;  
 newHead.prev = prev;  
  
 temp.prev = newTail;  
 newTail.next = temp;  
 }  
 size += c.size();  
 return true;  
 }  
 //Узнаём после какого элемента нам нужно вставлять коллекцию  
 private ListNode getElement(int index){  
 ListNode needEl=head;  
 for (int i=0;i<index;i++){  
 needEl=needEl.next;  
 }  
 return needEl;  
 }  
 //Удаляет все элементы из коллекции  
 @Override  
 public boolean removeAll(Collection<?> c) {  
 boolean logicValue = false;  
 for (Object o : c) {  
 logicValue = logicValue || remove(o);  
 }  
 return logicValue;  
 }  
 //Удаление элементов из списка, которые не содержатся в указанной коллекции.  
 @Override  
 public boolean retainAll(Collection<?> c) {  
 boolean mod = false;  
 if (head == null) {  
 return false;  
 }  
 ListNode data = null;  
 ListNode next = head;  
 while (next != null) {  
 if (!c.contains(next.vegetable)) {  
 if (data != null) {  
 data.next = next.next;  
 } else {  
 next = head.next;  
 head = head.next;  
 }  
 --size;  
 mod = true;  
 }  
 data = data != null ? data.next : head;  
 next = next.next;  
 }  
 return mod;  
 }  
 //Обнуляет список  
 @Override  
 public void clear() {  
 head=tail=null;  
 size=0;  
 }  
 @Override  
 public Vegetable get(int index) {  
 return getElement(index).vegetable;  
 }  
 //Устанавливает новый овощ по индексу, возвращая старый  
 @Override  
 public Vegetable set(int index, Vegetable element) {  
 ListNode data= getElement(index);  
 Vegetable oldVegetable = data.vegetable;  
 data.vegetable = element;  
 return oldVegetable;  
 }  
 //Добавляет элемент по индексу  
 @Override  
 public void add(int index, Vegetable element) {  
 if (index == size) {  
 add(element);  
 }  
 else {  
 ListNode data = getElement(index);  
 final ListNode prev = data.prev;  
 final ListNode newNode = new ListNode(element,data,prev);  
 data.prev = newNode;  
 if (prev == null)  
 head = newNode;  
 else  
 prev.next = newNode;  
 size++;  
 }  
 }  
 //Удаляет элемент по индексу и возвращает его  
 @Override  
 public Vegetable remove(int index) {  
 ListNode removeEll=head;  
 for(int i=0;i<index;i++){  
 removeEll=removeEll.next;  
 }  
 Vegetable v=removeEll.vegetable;//Определяем овощ  
 unlink(removeEll);  
 return v;  
 }  
 //Возвращает индекс первого вхождения указанного элемента  
 //в этом списке или -1, если этот список не содержит элемент.  
 @Override  
 public int indexOf(Object o) {  
 int index = 0;  
 for (ListNode data = head; data != null; data = data.next) {  
 if (o.equals(data.vegetable)) {  
 return index;  
 }  
 index++;  
 }  
 return -1;  
 }  
 //Возвращает индекс последнего вхождения указанного элемента  
 //в этом списке или -1, если этот список не содержит элемент.  
 @Override  
 public int lastIndexOf(Object o) {  
 int index = size-1;  
 for (ListNode data = tail; data != null; data = data.prev) {  
 if (o.equals(data.vegetable)) {  
 return index;  
 }  
 index--;  
 }  
 return -1;  
 }  
 @Override  
 public ListIterator<Vegetable> listIterator() {  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public ListIterator<Vegetable> listIterator(int index) {  
 throw new UnsupportedOperationException();  
 }  
  
 @Override  
 public List<Vegetable> subList(int fromIndex, int toIndex) {  
 throw new UnsupportedOperationException();  
 }  
}

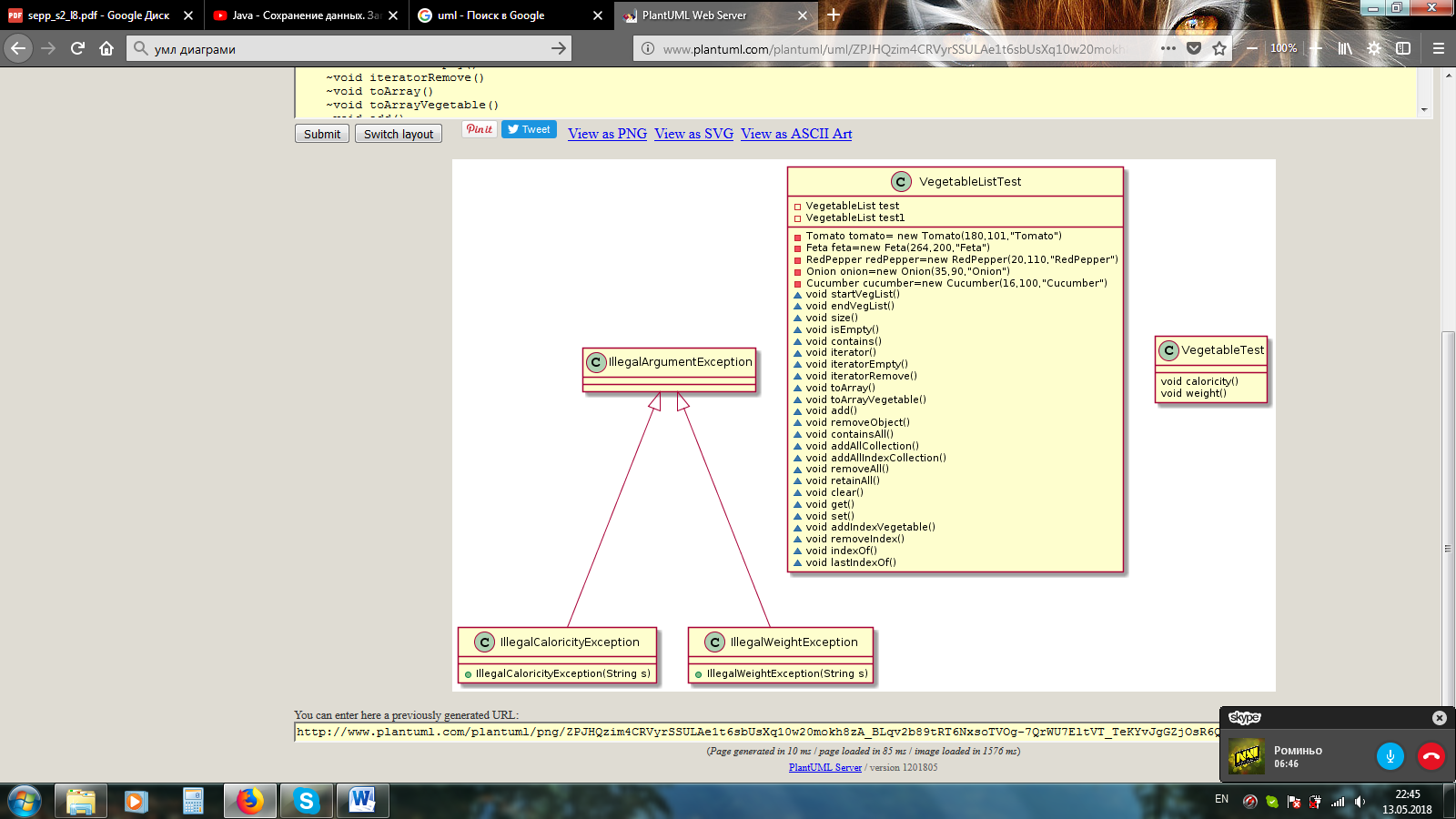
**VegetableListTest**

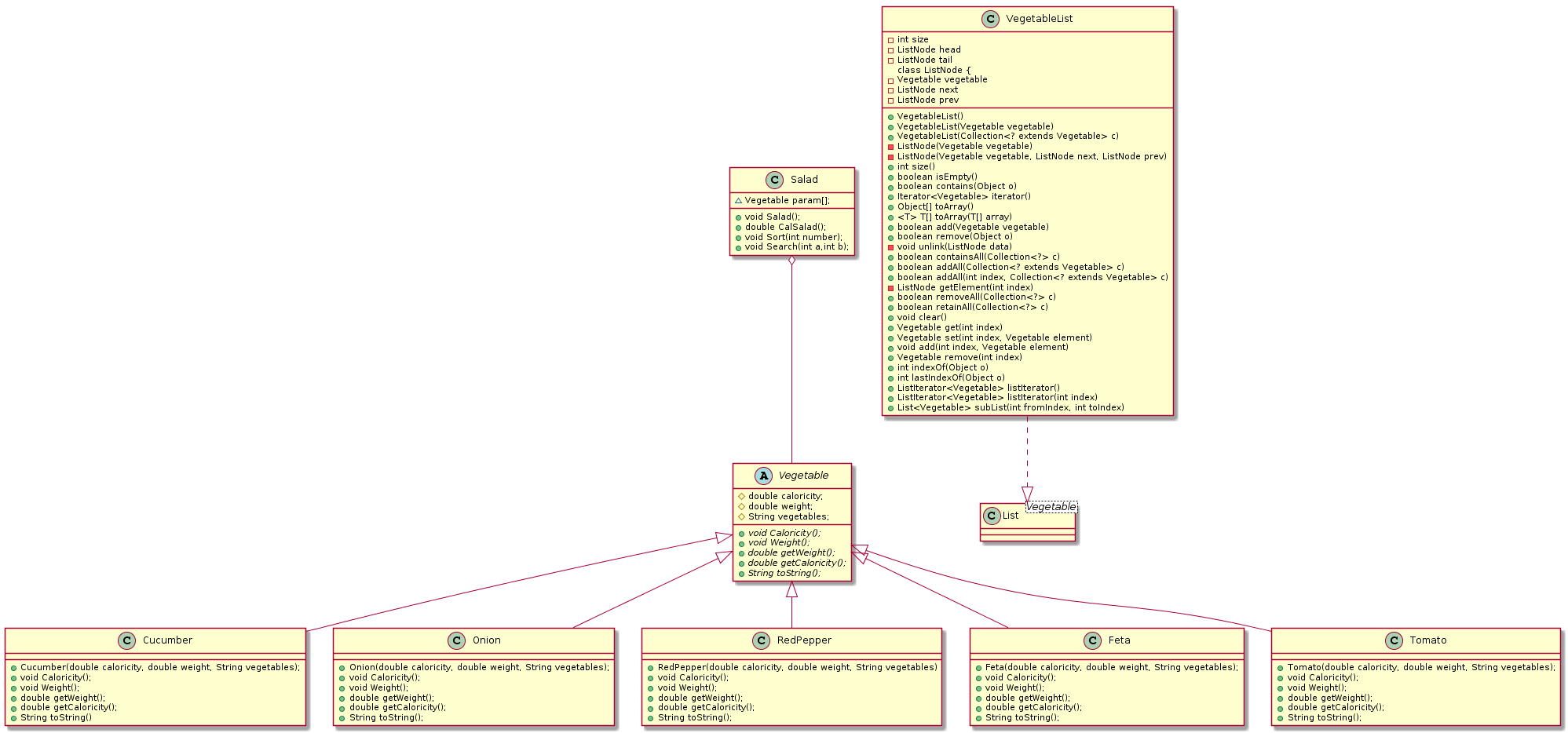
package com.company;  
  
import org.junit.jupiter.api.AfterEach;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import java.util.Iterator;  
import java.util.List;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class VegetableListTest {  
 private VegetableList test;  
 private VegetableList test1;  
 private Tomato tomato= new Tomato(180,101,"Tomato");  
 private Feta feta=new Feta(264,200,"Feta");  
 private RedPepper redPepper=new RedPepper(20,110,"RedPepper");  
 private Onion onion=new Onion(35,90,"Onion");  
 private Cucumber cucumber=new Cucumber(16,100,"Cucumber");  
  
 @BeforeEach  
 void startVegList(){  
 test=new VegetableList();  
 test.add(tomato);  
 test.add(feta);  
 test.add(redPepper);  
 test1=new VegetableList();  
 }  
  
 @AfterEach  
 void endVegList(){  
 test.clear();  
 test1.clear();  
 }  
  
 @Test  
 void size() {  
 *assertEquals*(3, test.size());  
 *assertEquals*(0, test1.size());  
 }  
  
 @Test  
 void isEmpty() {  
 *assertFalse*(test.isEmpty());  
 *assertTrue*(test1.isEmpty());  
 }  
  
 @Test  
 void contains() {  
 Onion onion1=new Onion(1,1,"onion1");  
 *assertFalse*(test1.contains(cucumber));  
 *assertFalse*(test1.contains(onion1));  
 *assertFalse*(test.contains(onion1));  
 *assertTrue*(test.contains(tomato));  
 test.add(onion1);  
 *assertTrue*(test.contains(onion1));  
 }  
  
 @Test  
 void iterator() {  
 Iterator iterator = test.iterator();  
 boolean a=false;  
 int i = 0;  
 while (iterator.hasNext()) {  
 i++;  
 iterator.next();  
 }  
 *assertEquals*(3, i);  
 if (iterator.next()== null){  
 a=true;  
 }  
 *assertTrue*(a);  
 }  
  
 @Test  
 void iteratorEmpty() {  
 Iterator iterator=test1.iterator();  
 *assertFalse*(iterator.hasNext());  
 }  
  
 @Test  
 void iteratorRemove() {  
 Iterator iterator = test.iterator();  
 try{iterator.remove();  
 *fail*("Remove not implemented.");}catch (UnsupportedOperationException a){*assertNotEquals*("",a.getMessage());}  
 }  
  
 @Test  
 void toArray() {  
 Object[] array = test.toArray();  
 *assertEquals*(test.get(0), array[0]);  
 *assertEquals*(test.size(), array.length);  
 }  
  
 @Test  
 void toArrayVegetable() {  
 Vegetable[] arr = test.toArray(new Vegetable[0]);  
 for (int i = 0; i < arr.length; i++) {  
 *assertEquals*(test.get(i), arr[i]);  
 }  
 }  
  
 @Test  
 void add() {  
 boolean a=test1.add(tomato);  
 *assertEquals*(1, test1.size());  
 *assertTrue*(a);  
 *assertEquals*(tomato,test1.get(0));  
 }  
  
 @Test  
 void removeObject() {  
 test1.add(onion);  
 int startSize=test1.size();  
 boolean a = test1.remove(onion);  
 *assertTrue*(a);  
 *assertEquals*(test1.size(), startSize-1);  
 }  
  
 @Test  
 void containsAll() {  
 boolean a= test.containsAll(test);  
 *assertTrue*(a);  
 }  
  
 @Test  
 void addAllCollection() {  
 List<Vegetable> newList = new VegetableList();  
 newList.add(feta);  
 newList.add(redPepper);  
 int startSizeTest = test.size();  
 int startSizeNewList = newList.size();  
 boolean a = test.addAll(newList);  
 *assertTrue*(a);  
 *assertEquals*(startSizeTest + startSizeNewList, test.size());  
 *assertEquals*(newList.get(0), test.get(test.size() - 2));  
 *assertEquals*(newList.get(1), test.get(test.size() - 1));  
 }  
  
 @Test  
 void addAllIndexCollection() {  
 List<Vegetable> newList = new VegetableList();  
 newList.add(feta);  
 newList.add(redPepper);  
 int startSizeTest = test.size();  
 int startSizeNewList = newList.size();  
 boolean a = test.addAll(0,newList);  
 *assertTrue*(a);  
 *assertEquals*(startSizeTest + startSizeNewList, test.size());  
 *assertEquals*(newList.get(0), test.get(0));  
 *assertEquals*(newList.get(1), test.get(1));  
 }  
  
 @Test  
 void removeAll() {  
 test.add(onion);  
 test1.add(onion);  
 int sizeTest = test.size();  
 int sizeTest1 = test1.size();  
 boolean a = test.removeAll(test1);  
 *assertEquals*(sizeTest - sizeTest1, test.size());  
 *assertTrue*(a);  
 }  
  
 @Test  
 void retainAll() {  
 test.add(onion);  
 test1.add(tomato);  
 test1.add(feta);  
 test1.add(redPepper);  
 int sizeTest1 = test1.size();  
 boolean a = test.retainAll(test1);  
 *assertEquals*(sizeTest1, test.size());  
 *assertTrue*(a);  
 }  
  
 @Test  
 void clear() {  
 test.clear();  
 test1.clear();  
 *assertTrue*(test.isEmpty());  
 *assertTrue*(test1.isEmpty());  
 }  
  
 @Test  
 void get() {  
 Vegetable vegetable = test.get(0);  
 *assertEquals*(tomato, vegetable);  
 }  
  
 @Test  
 void set() {  
 test.set(1, onion);  
 *assertEquals*(onion, test.get(1));  
 Vegetable vegetable=test.get(1);  
 *assertEquals*(vegetable, onion);  
 }  
  
 @Test  
 void addIndexVegetable() {  
 int sizeTest=test.size();  
 Vegetable vegetable=test.get(1);  
 test.add(1,onion);  
 *assertEquals*(sizeTest+1, test.size());  
 *assertEquals*(onion,test.get(1));  
 *assertEquals*(vegetable,test.get(2));  
 }  
  
 @Test  
 void removeIndex() {  
 test1.add(onion);  
 int startSize=test1.size();  
 Vegetable vegetable = test1.remove(0);  
 *assertEquals*(onion,vegetable);  
 *assertEquals*(test1.size(), startSize-1);  
 }  
  
 @Test  
 void indexOf() {  
 Vegetable vegetable= test.get(0);  
 *assertEquals*(0, test.indexOf(vegetable));  
 }  
  
 @Test  
 void lastIndexOf() {  
 test.add(tomato);  
 Vegetable vegetable= test.get(0);  
 *assertEquals*(3, test.lastIndexOf(vegetable));  
 }

**VegetableTest**

package com.company;  
  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
class VegetableTest {  
  
 @Test  
 void caloricity() {  
 *assertThrows*(IllegalCaloricityException.class, () -> new Cucumber(-9, 100, "Cucumber"));  
 }  
  
 @Test  
 void weight() {  
 *assertThrows*(IllegalArgumentException.class, () -> new Cucumber(120, 0, "Cucumber"));  
  
 }  
}

**Дiаграма класiв**





**Висновок**

Під час виконання завдання було модифіковано класи з попередніх лабораторних робіт (лабораторні роботи №5 та №6) таким чином, що обробка виключних ситуацій відбувається за допомогою стандартних засобів мови програмування Java. Створено 2 власних класи обробники виключних ситуацій. Написано JUnit-тести для перевірки працездатності усіх методів та виключних

ситуацій, всі тести пройшли перевірку, що підтверджує правильні алгоритми методів класу VegetableList. Всі початкові дані задаються у виконавчому методі. Код відповідає стандартам JCC.